EXPRESS MAIL LABEL NO.:

EV304738302US

IMPLEMENTING DEVICE SUPPORT IN A WEB-BASED ENTERPRISE APPLICATION

Martin Joseph Finnerty
Ming-Tao Liou
Blair T. Wheadon
Xia Liu
Ying-Chieh Lan

BACKGROUND OF THE INVENTION

[0002] As businesses increasingly rely on computers for their daily operations, managing the vast amount of business information generated and processed has become a significant challenge. Most large businesses have a wide variety of application programs managing large volumes of data that is entered and/or provided to users via many different types of input/output devices. As an example, in the banking industry, data may be provided using a variety of devices, such as an Automated Teller Machine (ATM) key entry pad, a magnetic card reader, a receipt printer, a passbook reader/writer, and so on. These input/output devices may be available via various types of networks and operating system platforms and often include a variety of devices produced by many different vendors. Each device typically is incompatible with the devices of other vendors.

[0003] Often, vendors of input/output devices provide their own application programming interfaces (APIs, sometimes referred to as application program interfaces) and/or command line utilities for using the specialized features of their own input/output devices, but these APIs and command line utilities are not compatible from vendor to vendor. Instead, these APIs and command line utilities typically communicate in a device-specific native language.

Furthermore, device communication standards have not been widely adopted, adding to the incompatibility problem.

[0004] One approach to communicating with a wide variety of types of devices is to modify an enterprise data management application that uses the features of these devices to communicate in the device-specific native language for each device. For example, to add a new type of input/output device to an enterprise data management system environment, the enterprise data management application often must be modified to call the device-specific APIs in addition to adding a new device driver or modifying an existing device driver. The software using the new device is tested by enterprise data management application vendor personnel and incorporated into a later release of the enterprise data management application.

[0005] This process for making new devices available has proven costly for vendors of enterprise data management applications, as support for each new type of device requires a new release, a costly and time-consuming project. Furthermore, the release process has proven untimely for the vendors of devices, as support for their products depends upon the release schedules for enterprise data management applications in which their products are used, slowing time to market for their devices. Typically, enterprise data management application vendors are not device vendors and would prefer a platform- and device-independent mechanism to request input/output services.

[0006] Another approach to simplify communication with a variety of different devices is provided by the Microsoft Windows Driver Model (WDM). WDM was introduced to allow driver developers to write device drivers that are source-code compatible across all Microsoft Windows operating systems. A Windows driver acts as an intermediary between an application program and a device driver. A Windows driver API is pre-defined to receive calls from application programs for services, and the Windows driver determines the appropriate device to provide the service and communicates with the selected device in that device's native language.

[0007] The Windows Driver Model, however, has failed to provide the flexibility needed by enterprise data management applications. Because the WDM API is pre-defined, all features provided by each device may not be supported. In addition, most API calls do not provide optional variables that can be set to take advantage of a particular device's features,

and no mechanism exists to easily modify messages sent to the WDM API. As a result, it may not be possible to take advantages of the particular features of a device without using the device-specific native language.

[0008] What is needed is the ability to dynamically add or modify support for a device in an enterprise data management application without the need to modify the enterprise data management application to communicate with the device.

SUMMARY OF THE INVENTION

[0009] The present invention relates to a method, system, application programming interface, and computer program product that enable enterprise data management application programs to request input/output services from a device manager controlling devices. The device manager operates in a heterogeneous environment including incompatible devices provided by multiple vendors. These requests for services, and responses from the device manager to the enterprise data management application program, are communicated in a markup language format, rather than in a device-specific native language.

[0010] An application programming interface is provided to communicate with the device manager in the markup language format. This structure enables the vendor of the enterprise data management application program to send requests for services, including both data and instructions to use specialized features of a device, without the need to change the enterprise data management application program to support device-specific native languages. The device manager can be easily modified to support new devices by adding new API commands to support specific features of those devices.

[0011] One feature of the invention includes a method for obtaining a request to provide a service, wherein the request conforms to a request format defined in a first language, such as a markup language. At least one device of a plurality of devices provides the service, and the method further includes identifying one of the devices to provide the service. The method further includes converting the request to a second request in a second language, such as a device-specific native language, wherein the second language is used for communication with the one device. The method further includes directing the second request to the one device.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] The present invention may be better understood, and its numerous objectives, features, and advantages made apparent to those skilled in the art by referencing the accompanying drawings.

[0013] Fig. 1A shows an example of a system including an enterprise application client directly communicating with a language-independent device manager in control of devices.

[0014] Fig. 1B shows an example of a system including an enterprise application client indirectly communicating with a language-independent device manager in control of devices via an intermediary enterprise application web server.

[0015] Fig. 2A shows an example of an initialization process in the system of Fig. 1A.

[0016] Fig. 2B shows an example of an initialization process in the system of Fig. 1B.

[0017] Fig. 3A shows an example of a print process in the system of Fig. 1A.

[0018] Fig. 3B shows an example of a print process in the system of Fig. 1B.

[0019] Fig. 4 is a use case diagram for the example systems shown in Figs. 1A and 1B.

[0020] Fig. 5A shows an example of a markup language request sent to the device manager of Fig. 1A or 1B for services provided by a device.

[0021] Fig. 5B shows an example of a markup language response sent by the device manager of Fig. 1A or 1B to a request for services, such as the request of Fig. 5A.

[0022] Fig. 5C shows an example of a markup language response indicating that the request for services was unsuccessful.

[0023] Fig. 5D shows an example of a markup language event occurring with respect to

one of the devices providing input/output services.

[0024] Fig. 6 is an example of a network environment in which the present invention can operate.

[0025] Fig. 7 is an example configuration of a computer system that can be used to operate the present invention.

[0026] The use of the same reference symbols in different drawings indicates similar or identical items.

DETAILED DESCRIPTION

[0027] For a thorough understanding of the subject invention, refer to the following Detailed Description, including the appended Claims, in connection with the above-described Drawings. Although the present invention is described in connection with several embodiments, the invention is not intended to be limited to the specific forms set forth herein. On the contrary, it is intended to cover such alternatives, modifications, and equivalents as can be reasonably included within the scope of the invention as defined by the appended Claims.

[0028] In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the invention. It will be apparent, however, to one skilled in the art that the invention can be practiced without these specific details.

[0029] References in the specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the invention. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment, nor are separate or alternative embodiments mutually exclusive of other embodiments. Moreover, various features are described which may be exhibited by some embodiments and not by others. Similarly, various requirements are described which may be requirements for some embodiments but not other embodiments.

Terminology

[0030] Communication with a device commonly involves use of a dynamic link library (DLL). A DLL is a collection of programs, any of which can be called when needed by another program that is running in the computer. A particular DLL program that enables the calling program to communicate with a specific device, such as a printer or scanner, is often packaged as a DLL file. DLL files that support specific device operation are known as device drivers.

[0031] The advantage of DLL files is that, because they are not loaded into random access memory (RAM) together with the main program, space is saved in RAM. When and if a DLL file is needed, then it is loaded and run. For example, as long as a user of Microsoft Word is editing a document, the printer DLL file does not need to be loaded into RAM. If the user decides to print the document, then the Word application causes the printer DLL file to be loaded and run.

[0032] As mentioned previously, typically device vendors will provide an application program interface (API) to communicate with a specific device. An API is the specific method prescribed by a computer operating system or by an application program by which a programmer writing an application program can make requests of the operating system or another application. An API can be contrasted with a graphical user interface or a command interface (both of which are direct user interfaces) as interfaces to an operating system or a program.

[0033] The term 'markup language' is used herein to describe a text-based language in which information can be encoded using structural features such as tags. SGML (Standard Generalized Markup Language) is one example of a standard for specifying a document markup language or tag set. Such a specification is itself a document type definition (DTD). SGML is not in itself a document language, but a description of how to specify one. SGML is metadata.

[0034] SGML is based on the idea that documents have structural and other semantic elements that can be described without reference to how such elements should be displayed. The actual display of such a document may vary, depending on the output medium and style

preferences. Some advantages of documents based on SGML are:

- Documents can be created by thinking in terms of document structure rather than appearance characteristics (which may change over time).

- Documents are more portable because an SGML compiler can interpret any document by reference to the document's document type definition (DTD).

- Documents originally intended for the print medium can easily be re-adapted for other media, such as the computer display screen.

[0035] The language that web browsers use, Hypertext Markup Language (HTML), is an example of an SGML-based language. There is a document type definition for HTML (and reading the HTML specification is effectively reading an expanded version of the document type definition). The markup languages described herein are not limited to SGML-based markup languages or HTML. The markup languages described herein need only provide the capability to define structure of the information in the document.

Introduction

[0036] The present invention relates to a method, system, application programming interface, and computer program product that enable enterprise data management application programs to request input/output services from a device manager controlling devices. The device manager operates in a heterogeneous environment including incompatible devices provided by multiple vendors. These requests, and responses from the device manager to the enterprise data management application program, are communicated in a markup language format, rather than in a device-specific native language. This structure enables the vendor of the enterprise data management application program to send requests for services, including both data and instructions to use specialized features of a device, without the need to change the enterprise data management application program to support device-specific native languages.

[0037] Fig. 1A shows an example of a system including an enterprise application client 110 in communication with an enterprise application web server 120 via communication link

122. Enterprise application client 110 uses input/output services provided by devices 130, with device 130A, device 130B, device 130C, and device 130D shown as representative devices. Each of devices 130A, 130B, 130C, and 130D has a respective device language-specific interface 132A, 132B, 132C, and 132D. These device language-specific interfaces 132 may be, for example, device native APIs. Enterprise application client 110 directly communicates with a language-independent device manager 140 in control of devices 130 to request input/output services. Devices 130, device language-specific interfaces 132, and device manager 140 form device controller system 134.

[0038] Device manager 140 can be considered to be language-independent because requests for input/output services are communicated by enterprise application client 110 using markup language messages, rather than the native languages for devices. These markup language messages can be communicated, for example, via Remote Procedure Calls (RPC) between computers on a network, Component Object Model (COM) method calls, or by direct API calls, as shown by communication 112A between enterprise application client 110 and device markup language API 150. Device manager may be implemented, for example, as a servlet program, as part of an enterprise application, or using other techniques known in the art. Within device manager 140, device markup language API 150 is an interface for receiving and sending messages in a markup language format. Once received by device manager 140 via device manager language API 150, these markup language messages can be provided via communication 152 to markup language parser 160. Markup language parser 160 parses the markup language message and provides the results to device manager 140, as shown by communication 142.

[0039] The results provided by markup language parser 160 are used by device manager 140 to communicate with devices 130 using respective device-specific native language(s) for each of devices 130, as represented by device language-specific interfaces 132. Each of device language-specific interfaces 132 can be considered to correspond to a dynamic link library (DLL) for a respective one of devices 130.

[0040] Enterprise application client 110 includes an applet 114 that is downloaded from enterprise application web server 120 to run within a browser program (not shown) used for Internet communication. Applet 114 can generate markup language messages to communicate with device manager 140, such as markup language messages to request

input/output services on behalf of enterprise application client 110. In one embodiment, applet 114 is implemented as a Java applet, although applet 114 represents any type of program that can be used by enterprise application client 110 to allow a user (person or another program) to request services.

[0041] Enterprise application client 110 also includes control 116, which is a control program that can run outside a browser program (not shown) in which applet 114 executes. For example, control 116 may send markup language messages generated by applet 114 to device manager 140. Control 116 may have access privileges for accessing the client computer system (not shown) on which enterprise application client 110 runs, whereas applet 114 may not. For example, control 116 may be able to access client computer system resources to identify an address for device manager 140, whereas an applet typically cannot obtain such information. Applet 114 and control 116 can communicate via script messages, as shown by communication 115. The use of applet 114 and control 116 is one example of an implementation of enterprise application client 110, but the invention is not limited to such an implementation. Any implementation in which an enterprise application client, such as enterprise application client 110, can communicate with a language-independent device manager, such as device manager 140, is within the scope of the invention. Another example implementation of enterprise application client 110 is provided in Fig. 1B, as described below.

[0042] Fig. 1B shows an example of a system including an enterprise application client indirectly communicating with a language-independent device manager in control of devices via an intermediary enterprise application web server. The functionality of enterprise application client 110, devices 130, and device manager 140 is essentially the same as described with reference to Fig. 1A. However, in this embodiment, enterprise application client 110 sends requests for input/output services via enterprise application web server 120 rather than directly to device manager 140. Instead of having a control program with access privileges on the client computer system running enterprise application client 110, applet 114 communicates markup language messages, as shown by communication 117, to server communication module 122C.

[0043] Server communication module 122C communicates with proxy server 122S via communication link 122. Communication via communication link 122 can be implemented

using RPC, COM, hypertext transfer protocol (HTTP), or other communication protocols. Proxy server 122S in turn provides messages, as shown in client/server communication 123, to enterprise application web server 120. Communication 123 can be accomplished, for example, using direct API calls, COM, or other inter-process communication techniques. Enterprise web application web server 120 then communicates markup language messages to device manager 140 via device markup language API 150, as shown in communication 112B. Communication 112B can be accomplished using RPC, COM, or direct API calls.

[0044] While Fig. 1B shows enterprise application client 110 as generating a request for service from device manager 140, it is possible that enterprise application server 120 may recognize the need to generate a request for service without enterprise application client 110 making the request. Enterprise application server 120 can then generate and send the appropriate markup language messages to make the request.

[0045] Fig. 2A shows an example of an initialization process in the system of Fig. 1A, where enterprise application client 110 communicates directly with device manager 140. Encircled guide numbers are provided in Figs. 2A through 3B to aid the reader. In action 2A.1, enterprise application client 110 is loaded. In action 2A.2, enterprise application client 110 invokes an initialization method, which initializes applet 114. In action 2A.3, applet 114 generates markup language messages to request device manager 140 for information about attached devices that are available for use by enterprise application client 110. In action 2A.4, applet 114 provides the markup language messages to request information about attached devices to control 116. Control 116 may have an API with which applet 114 must communicate, and applet 114 may need to process the markup language messages into a format that is compatible with the API for control 116. If an API for control 116 exists, control 116 converts the messages from the API format into the markup language message format for communication to device manager 140.

[0046] As noted above, control 116 has the ability to determine the address of device manager 140 and to communicate messages to device manager 140 on behalf of applet 114. This communication can be accomplished in several ways. For example, control 116 may send a message directly to device manager 140 using the address. Alternatively, control 116 may place a message into a queue that is periodically checked by device manager 140 to obtain messages, and device manager 140 may obtain the message from the queue. Other

ways of communicating messages between enterprise application client 110 and device manager 140 are also within the scope of the invention.

[0047] In action 2A.5, control 116 sends the markup language messages generated by applet 114 to device manager 140 by calling a function of device markup language API 150. In action 2A.6, device manager 140 processes the request for information about attached devices. Processing the request may involve providing the markup language messages to markup language parser 160 of Fig. 1A (not shown here), obtaining the results of parsing the markup language messages, and initiating communication with devices 130 (not shown) via device language-specific interfaces 132 to identify the available attached devices. In action 2A.7, device manager 140 uses device markup language API 150 to provide a response to the request for information about attached devices to control 116. In action 2A.8, control 116 provides the response received from device manager 140 to applet 114. In action 2A.9, applet 114 processes the response and stores the attached device information for use in future input/output service requests.

[0048] Fig. 2B shows an example of a similar initialization process in the system of Fig. 1B, where enterprise application client 110 communicates indirectly with device manager 140 via enterprise application web server 120. In action 2B.1, enterprise application client 110 is loaded. In action 2B.2, enterprise application client 110 invokes an initialization method, which initializes applet 114. In action 2B.3, applet 114 generates markup language messages to request device manager 140 for information about attached devices that are available for use by enterprise application client 110. Up until this point, all actions performed by enterprise application client 110 are the same as those described with reference to Fig. 1A. However, in action 2B.4, applet 114 provides the markup language messages to request information about attached devices to server communication module 122C rather than to a control, such as control 116 of Fig. 2B. Server communication module 122C may provide Remote Procedure Call (RPC) services, COM services, or other types of services for communication with enterprise application web server 120.

[0049] In action 2B.5, server communication module 122C sends the markup language messages generated by applet 114 to proxy server 122S, which provides corresponding RPC, COM, or other communication services to communicate with server communication module 122C. In action 2B.6, proxy server 122S provides the messages received to enterprise

application web server 120. In action 2B.7, enterprise application web server 120 sends the markup language messages to device manager 140 by calling a function of device markup language API 150. In action 2B.8, device manager 140 processes the request for information about attached devices. Processing the request may involve providing the markup language messages to markup language parser 160 of Fig. 1B (not shown here), obtaining the results of parsing the markup language messages, and initiating communication with devices 130 (not shown) via device language-specific interfaces 132 to identify the available attached devices.

[0050] In action 2B.9, device manager 140 uses device markup language API 150 to provide a response to the request for information about attached devices to enterprise application web server 120. In action 2B.10, enterprise application web server 120 provides the response received from device manager 140 to proxy server 122S. In action 2B.11, proxy server 122S sends the response including the attached devices information to server communication module 122C within enterprise application client 110. In action 2B.12, server communication module 122C provides the response to the original requester, applet 114. In action 2B.13, applet 114 processes the response and stores the attached device information for use in future input/output service requests.

[0051] Fig. 3A shows an example of a print process in the system of Fig. 1A, where enterprise application client 110 communicates directly with device manager 140. In action 3A.1, applet 114 receives a user request to print. The user request is typically provided via another program, such as a user interface for enterprise application client 110, that constructs the request in accordance with device language markup API. However, it is within the scope of the invention that a user provides the request via a command line interface. In action 3A.2, applet 114 generates a markup language print request to request printing services from device manager 140. In action 3A.3, applet 114 provides the markup language print request to control 116 via a pre-defined script function call.

[0052] In action 3A.4, control 116 sends the markup language print request generated by applet 114 to device manager 140 by calling a function of device markup language API 150. In action 3A.5, device manager 140 processes the print request. Processing the print request may involve providing the print request to markup language parser 160 of Fig. 1A (not shown here), obtaining the results of parsing the print request, selecting a device of devices 130 of Fig. 1A (not shown here) to fulfill the print request, translating the print request into a native

language for the selected device, and initiating communication with the selected device (not shown) via that device's respective device language-specific interfaces 132 to communicate the print request.

[0053] In action 3A.6, device manager 140 uses device markup language API 150 to provide a response to the print request to control 116. The response to the print request may include a status of whether the print request was successfully completed. In action 3A.7, control 116 provides the response received from device manager 140 to applet 114 via a pre-defined script function call. In action 3A.8, applet 114 processes the response including the status of the print request. For example, if the response indicates that the print request was unsuccessful, applet 114 may send another print request to device manager 140.

[0054] Fig. 3B shows an example of a print process in the system of Fig. 1B, where enterprise application client 110 communicates indirectly with device manager 140 via enterprise application web server 120. In action 3B.1, applet 114 receives a user request to print. In action 3B.2, applet 114 generates a markup language print request to request printing services from device manager 140. In action 3B.3, applet 114 provides the markup language print request to server communication module 122C.

[0055] In action 3B.4, server communication module 122C sends the markup language print request generated by applet 114 to proxy server 122S. In action 3B.5, proxy server 122S provides the markup language print request to enterprise application web server 120. Enterprise application web server 120 then sends the markup language print request to device manager 140 by calling a function of device markup language API 150 in action 3B.6. In action 3B.7, device manager 140 processes the markup language print request. Processing the print request may involve providing the print request to markup language parser 160 of Fig. 1B (not shown here), obtaining the results of parsing the print request, selecting a device of devices 130 of Fig. 1B (not shown here) to fulfill the print request, translating the print request into a native language for the selected device, and initiating communication with the selected device (not shown) via that device's respective native API of device language-specific interfaces 132 to communicate the print request.

[0056] In action 3B.8, device manager 140 uses device markup language API 150 to provide a response to the print request to enterprise application web server 120. The response

to the print request may include a status of whether the print request was successfully completed. In action 3B.9, proxy server 122S obtains the response from enterprise application web server 120. In action 3B.10, proxy server 122S provides the response received from device manager 140 to server communication module 122C within enterprise application client 110. In action 3B.11, enterprise application client 110 sends the response to the print request, including the status, to applet 114. In action 3B.12, applet 114 processes the response including the status of the print request. For example, if the response indicates that the print request was unsuccessful, applet 114 may send another print request to device manager 140.

[0057] In the examples described above with reference to Figs. 2A through 3B, device manager 140 performs several functions and can be considered to serve as a module, means, or instructions for performing each of those functions. For example, device manager 140 obtains a request to provide a service from enterprise application client 110 and can be considered to be an obtaining module, means, and/or instructions. Device manager 140 also performs the functions of identifying or selecting one device to provide the service; converting the request to a second request in a second language, wherein the second language is a device-specific native language used for communication with the one device; and directing the second request to the one device.

[0058] Fig. 4 is a use case diagram for the example systems shown in Figs. 1A and 1B. In this example, the user requests to print a document. Classes for objects involved in processing the print request are shown, with a lower-case 'c' used at the end of the reference numeral from Fig. 1A or 1B to indicate that a class providing a description for a related object is depicted. For example, the class for applet 114 is shown as applet 114c, and the class for control 116 is shown as control 116c. The print request is obtained by an object, such as applet 114, which instantiates applet class 114c within enterprise application client 110.

[0059] Objects instantiating applet class 114c can be configured to use objects instantiating a control class, such as control 116 instantiating control class 116c, and/or objects instantiating a server communication module class, such as server communication module 122C instantiating server communication module class 122Cc. Objects instantiating control class 116c can be configured to communicate with a device manager object, such as

device manager 140 instantiating a device manager class 140c, using device markup language API 150 (not shown). By having access to an object instantiating device manager class 140c, an object instantiating control class 116c can communicate with device controller system 134 and device language-specific interfaces 132 to access devices 130. Server communication module class 122Cc is configured to use a proxy server object, instantiating proxy server class 122Sc.

[0060] The markup language messages communicated between objects can be considered to include a header, made up of instructions, and a payload, which includes data upon which the instructions are to be performed. Markup language messages can be configured to use specific features of particular devices by specifying an optional variable in the instructions and providing a value for the optional variable in the payload. Device markup language API 150 of Figs. 1A and 1B can define a messaging protocol for communicating with device manager 140. For example, device markup language API 150 can include a request definition for a command providing a request for service to device manager 140, a response definition for a response format in which a response to the request is provided, and an initialization command for initializing prior to providing a request for service.

[0061] By providing a markup language message format, clients of device manager 140 can provide the optional variables and values as part of a configurable request. Markup language parser 160 can be configured to recognize those optional variables, and device manager 140 can convert the request to a device language-specific interface commands. Applications can take advantage of special features provided by a specific device without the need to modify source code of the applications.

[0062] Figs. 5A through 5D show examples of different types of markup language messages used for communication within the system described herein. Fig. 5A shows an example of a request 510 sent to the device manager 140 of Fig. 1A or 1B for services provided by a device, such as one of devices 130. Request 510 is delimited by the <deviceRequest> and </deviceRequest> tags at the top and bottom of request 510. Request 510 includes instructions 512, delimited by the <instruction> and </instruction> tags, and data 514, delimited by the <data> and </data> tags. In this example, instructions 512 specify four parameters: a request ID, delimited by the <requestID> and </requestID> tags and having a value of 'request1;' a requester, delimited by the <requester> and </requester> tags

and having a value of 'client1;' a device request, delimited by the <deviceRequest> and </deviceRequest> tags and having a value of 'Printer1;' and response required, delimited by the <responseRequired> and </responseRequired> tags and having a value of 'Y.' Data 514 provides names and values for N data fields upon which the request is to be performed.

[0063] Fig. 5B shows an example of a successful response sent by device manager 140 of Fig. 1A or 1B to a request for services, such as request 510 of Fig. 5A. Response 520 is delimited by the <deviceResponse> and </deviceResponse> tags at the top and bottom of response 520. Response 520 includes instructions 522, delimited by the <instruction> and </instruction> tags, and data 524, delimited by the <data> and </data> tags. In this example, instructions 522 specify two parameters: a request ID for a request to which the response applies, delimited by the <requestID> and </requestID> tags and having a value of 'request1;' and a requester to whom the response is to be provided, delimited by the <requester> and </requester> tags and having a value of 'client1.' Data 524 provides a name of a function, "print," and a value for an actionResponse data field, "SUCCESS," indicating that "SUCCESS" should be printed.

[0064] Fig. 5C shows an example of an unsuccessful response sent by device manager 140 of Fig. 1A or 1B to a request for services, such as request 510 of Fig. 5A. Response 530 is delimited by the <deviceResponse> and </deviceResponse> tags at the top and bottom of response 530. Response 530 includes instructions 532, delimited by the <instruction> and </instruction> tags, and data 534, delimited by the <data> and </data> tags. In this example, instructions 532 specify two parameters: a request ID for a request to which the response applies, delimited by the <requestID> and </requestID> tags and having a value of 'request1;' and a requester to whom the response is to be provided, delimited by the <requester> and </requester> tags and having a value of 'client1.' Data 534 provides a name of a function, "print," and a value for an actionResponse data field, "ERROR," indicating that "ERROR" should be printed. Data 534 further includes error information, including an error number field, delimited by the <errorNo> and </errorNo> tags and having a value of ERR_12345, and a message, delimited by the <message> and </message> tags and having a value of "Printer out of paper."

[0065] Figs. 5A through 5C have described the markup language format for requests and responses sent between enterprise application client 110 and device manager 140. However,

not all communication between device manager 140 and devices 130 is initiated by the enterprise application client 110. For example, data may be entered by swiping a card through a magnetic card reader device. When device manager 140 detects that a card swipe event has occurred, device manager 140 encodes the event as a markup language message. Applications can register with device manager 140 to receive notification of particular events of interest.

[0066] Fig. 5D shows an example of a device event 540. Event 540 is delimited by the <Event> and </Event> tags at the top and bottom of event 540. Event 540 includes instructions 542, delimited by the <instruction> and </instruction> tags, and data 544, delimited by the <data> and </data> tags. In this example, instructions 542 specify five parameters: an event ID for the event, delimited by the <eventID> and </eventID> tags and having a value of 'event1;' a device name, delimited by the <deviceName> and </deviceName> tags and having a value of 'deviceName;' a device type, delimited by the <deviceType> and </deviceType> tags and having a value of 'cardSwipeDevice1;' a device event, delimited by the <deviceEvent> and </deviceEvent> tags and having a value of 'cardSwipe;' and response required, delimited by the <responseRequired> and </responseRequired> tags and having a value of 'N.' Data 544 provides a name for a card number field, "card_number," and a value for the card number field, 'XXXXXXXXXX.'

[0067] The above-described embodiments of the invention can be implemented in a variety of computing and networking environments. An example networking environment in which the present invention can operate is described below with reference to Fig. 6, and an example computing environment that can be used to implement the invention is described below with reference to Fig. 7.

An Example Networking and Computing Environment

[0068] Fig. 6 is a block diagram illustrating a network environment in which a system according to the present invention may be practiced. As is illustrated in Fig. 6, network 600, such as a private wide area network (WAN) or the Internet, includes a number of networked servers 610(1)-(N) that are accessible by client computers 620(1)-(N). Communication between client computers 620(1)-(N) and servers 610(1)-(N) typically occurs over a publicly accessible network, such as a public switched telephone network (PSTN), a DSL connection, a cable modem connection or large bandwidth trunks (e.g., communications channels

providing T1 or OC3 service) or wireless link. Client computers 620(1)-(N) access servers 610(1)-(N) through, for example, a service provider. This might be, for example, an Internet Service Provider (ISP) such as America On-Line™, Prodigy™, CompuServe™ or the like. Access is typically had by executing application specific software (e.g., network connection software and a browser) on the given one of client computers 620(1)-(N).

[0069] One or more of client computers 620(1)-(N) and/or one or more of servers 610(1)-(N) may be, for example, a computer system of any appropriate design, in general, including a mainframe, a mini-computer or a personal computer system. One such example computer system, discussed in terms of client computers 620(1)-(N), is shown in detail in Fig. 7.

[0070] Fig. 7 depicts a block diagram of a computer system 710 suitable for implementing the present invention, and as an example of one or more of client computers 620(1)-(N) of Fig. 6. Computer system 710 includes a bus 712 which interconnects major subsystems of computer system 710 such as a central processor 714, a system memory 716 (typically RAM, but which may also include ROM, flash RAM, or the like), an input/output controller 718, an external audio device such as a speaker system 720 via an audio output interface 722, an external device such as a display screen 724 via display adapter 726, serial ports 728 and 730, a keyboard 732 (interfaced with a keyboard controller 733), a storage interface 734, a floppy disk drive 736 operative to receive a floppy disk 738, and a CD-ROM drive 740 operative to receive a CD-ROM 742. Also included are a mouse 746 (or other point-and-click device, coupled to bus 712 via serial port 728), a modem 747 (coupled to bus 712 via serial port 730) and a network interface 748 (coupled directly to bus 712).

[0071] Bus 712 allows data communication between central processor 714 and system memory 716, which may include both read only memory (ROM) or flash memory (neither shown), and random access memory (RAM) (not shown), as previously noted. The RAM is generally the main memory into which the operating system and application programs are loaded and typically affords at least 64 megabytes of memory space. The ROM or flash memory may contain, among other code, the Basic Input-Output system (BIOS) which controls basic hardware operation such as the interaction with peripheral components. Applications resident with computer system 710 are generally stored on and accessed via a computer readable medium, such as a hard disk drive (e.g., fixed disk 744), an optical drive (e.g., CD-ROM drive 740), floppy disk unit 736 or other storage medium. Additionally, applications may be in the form of electronic signals modulated in accordance with the application and data communication technology when accessed via network modem 747 or

interface 748.

[0072] Storage interface 734, as with the other storage interfaces of computer system 710, may connect to a standard computer readable medium for storage and/or retrieval of information, such as a fixed disk drive 744. Fixed disk drive 744 may be a part of computer system 710 or may be separate and accessed through other interface systems. Modem 747 may provide a direct connection to a remote server via a telephone link or to the Internet via an internet service provider (ISP). Network interface 748 may provide a direct connection to a remote server via a direct network link to the Internet via a POP (point of presence). Network interface 748 may provide such connection using wireless techniques, including digital cellular telephone connection, general packet radio service (GPRS) connection, digital satellite data connection or the like.

[0073] Many other devices or subsystems (not shown) may be connected in a similar manner (e.g., bar code readers, document scanners, digital cameras and so on). Conversely, it is not necessary for all of the devices shown in Fig. 7 to be present to practice the present invention. The devices and subsystems may be interconnected in different ways from that shown in Fig. 7. The operation of a computer system such as that shown in Fig. 7 is readily known in the art and is not discussed in detail in this application. Code to implement the present invention may be stored in computer-readable storage media such as one or more of system memory 716, fixed disk 744, CD-ROM 742, or floppy disk 738. Additionally, computer system 710 may be any kind of computing device, and so includes personal data assistants (PDAs), network appliances, X-window terminals or other such computing devices. Computer system 710 also supports a number of Internet access tools, including, for example, an HTTP-compliant web browser having a script interpreter, such as a JavaScript interpreter.

[0074] As explained above, the invention described herein can be implemented in a variety of computing and networking environments and is designed to facilitate communication between heterogeneous, incompatible devices and computer systems provided by multiple vendors. The invention enables enterprise data management application programs to request input/output services from a device manager controlling devices without requiring device-specific knowledge. Requests for services, and responses from the device manager to the enterprise data management application program, are communicated in a markup language format, rather than in a device-specific native language. This structure enables the vendor of the enterprise data management application program to send requests for services, including both data and instructions to use specialized features of a device,

without the need to change the enterprise data management application program to support device-specific native languages.

[0075] By using a markup language format for communication, integration of new devices is straightforward. Testing for new devices involves ensuring that the message formats conform to the device markup language API request and response formats. Optional variables and values can be specified to take advantage of specific features of a particular device without requiring modification of software for other devices.

Other Embodiments

[0076] The present invention is well adapted to attain the advantages mentioned as well as others inherent therein. While the present invention has been depicted, described, and is defined by reference to particular embodiments of the invention, such references do not imply a limitation on the invention, and no such limitation is to be inferred. The invention is capable of considerable modification, alteration, and equivalents in form and function, as will occur to those ordinarily skilled in the pertinent arts. The depicted and described embodiments are examples only, and are not exhaustive of the scope of the invention.

[0077] The foregoing detailed description has set forth various embodiments of the present invention via the use of block diagrams, flowcharts, and examples. It will be understood by those within the art that each block diagram component, flowchart step, operation and/or component illustrated by the use of examples can be implemented, individually and/or collectively, by a wide range of hardware, software, firmware, or any combination thereof.

[0078] The present invention has been described in the context of fully functional computer systems; however, those skilled in the art will appreciate that the present invention is capable of being distributed as a program product in a variety of forms, and that the present invention applies equally regardless of the particular type of signal bearing media used to actually carry out the distribution. Examples of signal bearing media include recordable media such as floppy disks and CD-ROM, transmission type media such as digital and analog communications links, as well as media storage and distribution systems developed in the future.

[0079] The above-discussed embodiments may be implemented by software modules that perform certain tasks. The software modules discussed herein may include script, batch, or other executable files. The software modules may be stored on a machine-readable or computer-readable storage medium such as a disk drive. Storage devices used for storing software modules in accordance with an embodiment of the invention may be magnetic floppy disks, hard disks, or optical discs such as CD-ROMs or CD-Rs, for example. A storage device used for storing firmware or hardware modules in accordance with an embodiment of the invention may also include a semiconductor-based memory, which may be permanently, removably or remotely coupled to a microprocessor/memory system. Thus, the modules may be stored within a computer system memory to configure the computer system to perform the functions of the module. Other new and various types of computer-readable storage media may be used to store the modules discussed herein.

[0080] The above description is intended to be illustrative of the invention and should not be taken to be limiting. Other embodiments within the scope of the present invention are possible. Those skilled in the art will readily implement the steps necessary to provide the structures and the methods disclosed herein, and will understand that the process parameters and sequence of steps are given by way of example only and can be varied to achieve the desired structure as well as modifications that are within the scope of the invention. Variations and modifications of the embodiments disclosed herein can be made based on the description set forth herein, without departing from the scope of the invention. Consequently, the invention is intended to be limited only by the scope of the appended claims, giving full cognizance to equivalents in all respects.